JC978 U.S. PTO 09/955913 09/18/01

# 2X Multiplier

## Approval

| Who | Group | Date Approved |
| --- | --- | --- |
| Gary Lai | Originator | |
| Paige Kolze | Hardware | |
| Hung Nguyen | Hardware | |
| Brad Taylor | Systems | |
| Jack Greenbaum | Software | |
| Alan Wessel | Marketing | |
| | | |

## Revision History

| Revision | Date | Author | Description |
| --- | --- | --- | --- |
| 1.0 | 4/18/01 | Gary Lai | Original (taken from old Ver 1.2 3/28/2001). |
| | | | |
| | | | |

# 1 EXTENDED MULTIPLIER

## 1.1 Purpose

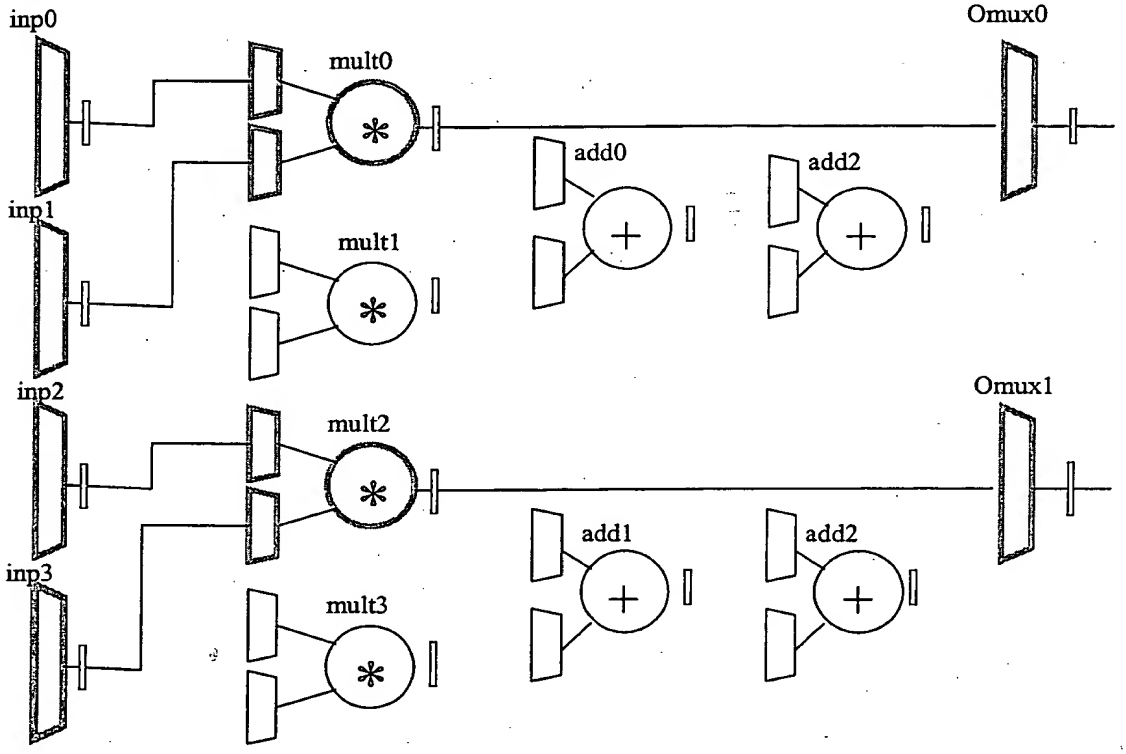
The purpose of this extended multiplier, is to build more functionality into the existing multiplier, to optimize it for the expected customer applications. This will be accomplished, by adding 2 additional multipliers per tile, and 4 additional adders. These will all be added to the current multiplier block, in essence creating a large functional unit that is capable of significant processing.

In order to reduce the overhead in this effort, it has been decided to use the same input muxes and output muxes as the current multipliers. This means that we have a block that has 4 * 32-bit inputs and 2 * 32-bit outputs. Thus in order to take advantage of having 4 multipliers, the inputs somehow have to be shared between them, and the outputs need to be shared as well. The sharing of the inputs is generally accomplished by packing the inputs into the high and low halves of the input words. The output sharing is accomplished by either accumulating the results through the new adders, or packing the results into the output registers.

Finally, for backward compatibility, we need to provide for the case where the new features are bypassed. This is accomplished by making it so that the input and output muxes are selected in the same fashion for the old bits, and the new bits (when defaulted to 0) do not effect the circuit. The opcode is also designed so that the default case of all 0's causes the mutlipliers to behave as before.

## 2 IMPLEMENTATION

The XMULT or "extended multiplier" builds on the existing multiplier block by maintaining the existing I/O structure. It adds 2 additional multipliers and 4 adders to create a structure which can be configured as for a variety of different functions. A simplified diagram of the multiplier is illustrated below, along with a table of the target opcodes. The wiring pattern illustrates a backwards compatible mode. Components in bold illustrate the existing multiplier. Each adder or multiplier has 2 input muxes and an optional output register.




| opcode | latency | function |
| --- | --- | --- |
| 2MULT | 2 | current multiplier mode |
| 4ADD32 | 3 | sum of 4 32-bit inputs |
| 4ADD16 | 4 | sum of 4 sets of packed 16 bit inputs |
| 4MULT | 2 | 4 independent multipliers with 16-bit packed inputs and outputs |
| 4MULTSUM | 4 | 4 multipliers with 16-bit packed inputs and outputs summed together in tree |
| 4MULT2SUM | 3 | 4 multipliers with 16-bit packed inputs and outputs summed together in tree |
| 4FIR | N/A | 4 multipliers with 16-Bit packed coefficients, a single 16-Bit data input, a 32-bit accumulation input and a 32-Bit outputs accumulated in cascade with pipeline registers between accumulators |
| CMULT | 4 | 16-Bit packed complex multiply with 32-Bit IQ accumulation input,output |
| CMULT16 | 3 | 16-Bit packed complex multiply with FFT butterfly adders with complex input, output |

OPERATOR MUX selects by OPCODE

| MUX | n | OPCODE | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 2MULT | 4ADD32 | 4ADD16 | 4MULT | 4MULT+ | 4MULT2+ | 4FIR | CMULT16 | CMULT |
| mult0-a | | i0[31:16] | | | i0[31:16] | i0[31:16] | i0[31:16] | i2[15:0] | i0[31:16] | i0[31:16] |
| | | i0[15:0] | | | | | | | | |
| mult0-b | | i1[31:8] | | | i1[31:16] | i1[31:16] | i1[31:16] | i1[31:16] | i1[31:16] | i1[31:16] |
| | | i1[31:16] | | | | | | | | |
| | | i1[15:0] | | | | | | | | |
| mult1-a | | | | | i0[15:0] | i0[15:0] | i0[15:0] | i2[15:0] | i0[15:0] | i0[15:0] |
| mult1-b | | | | | i1[15:0] | i1[15:0] | i1[15:0] | i1[15:0] | i1[15:0] | i1[15:0] |
| mult2-a | | i2[31:16] | | | i2[31:16] | i2[31:16] | i2[31:16] | i2[15:0] | i0[31:16] | i0[31:16] |
| | | i2[15:0] | | | | | | | | |
| mult2-b | | i3[31:8] | | | i3[31:16] | i3[31:16] | i3[31:16] | i3[31:16] | i1[15:0] | i1[15:0] |
| | | i3[31:16] | | | | | | | | |
| | | i3[15:0] | | | | | | | | |
| mult3-a | | | | | i2[15:0] | i2[15:0] | i2[15:0] | i2[15:0] | i0[15:0] | i0[15:0] |
| mult3-b | | | | | i3[15:0] | i3[15:0] | i3[15:0] | i3[15:0] | i1[31:16] | i1[31:16] |
| add0-a | 2 | | i0[31:0] | i0[31:0] | | m0[31:0] | m0[31:0] | i0[31:0] | m0[31:0] | m0[31:0] |
| add0-b | 3 | | i1[31:0] | i1[31:0] | | m1[31:0] | m1[31:0] | m1[31:0] | ~m1 + 1 | ~m1 + 1 |
| add1-a | 3 | | i2[31:0] | i2[31:0] | | m2[31:0] | m2[31:0] | a2[31:0] | m2[31:0] | m2[31:0] |
| add1-b | 2 | | i3[31:0] | i3[31:0] | | m3[31:0] | m3[31:0] | m3[31:0] | m3[31:0] | m3[31:0] |
| add2-a | 2 | | a0[31:0] | a0[31:0] | | a0[31:0] | | m0[31:0] | | a0[31:0] |
| add2-b | 3 | | a1[31:0] | a1[31:0] | | a1[31:0] | | a0[31:0] | | i2[31:0] |
| add3-a | 3 | | | a2[31:16] | | | | a1[31:0] | 32'h0 | a1[31:0] |
| add3-b | | | | a2[15:0] | | | | m2[31:0] | i3[31:0] | i3[31:0] |
| omux0 | 6 | m0[31:0] | a2[31:0] | a2[31:0] | m0[31:16] | a2[31:0] | a0[31:0] | | a0[31:16] | a2[31:0] |
| | | | | | m1[31:16] | | | | a1[31:16] | |
| omux1 | 6 | m2[31:0] | a2[Cout] | a3[31:0] | m2[31:16] | a2[Cout] | a1[31:0] | a3[31:0] | a3[31:0] | a3[31:0] |
| | | | | | m3[31:16] | | | | | |

i0-i3 = input mux
m0-m3=mults
a0-a3=adder
-a = operand a
-b= operand b

Carry of 32-bit ADD operation is not brought out unless explicitly specified in this document. The precision of the ADD operation right after the multipliers is not lost due to the duplicate sign bit in the result of the multipliers. For any other additions, it the user's responsibility to avoid the event of a overflow. The user might also use the shift-down operation in the result of the adders to reduce the loss of precision.

## 2.1 Bit file encoding

There are 10 bits in the CSMMULT that are not used. These are 'RESERVED' fields, as well as the mult_h lsm wen, and the mult_h lsm dynamic mode bits. The wen is not used, as lsm3 is never used as a write lsm unless it is connected to at least one other lsm. The write enable is routed with the write address, and the multiplier cannot generate the write address. The dynamic mode bits are routed with the write data, and the multiplier cannot generate write data. Thus, neither of these fields is meaningful in the CSMMULT.

The multiplier a input mux select is named muxafghsel, which is currently a 1 bit field. We will extend this to 2 bits for both mutlipliers, at the cost of 2 CSM bits. The output select is named muxmultlsmsel, which is currently a 2 bit field. We will extend this to 3 bits for both mutlipliers, at the cost of 2 CSM bits. We will add a 5 bit opcode, which will essentially be shared, which will therefore cost 5 CSM bits. One of the remaining 2 bits will be used to selectively shift all of the multiplier results up by one bits, thereby normalizing off the redundant sign bit. The other remaining bit will be used to selectively shift the adder outputs down by one bit, in order to normalize the adder results. Thus, all of the available CSMMULT bits will be utilized by the new design.

Opcodes (new 4 bit CSMMULT field)

| Name | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Multbypass[3:0] | Adder s16bit[3:0] | Adder bypass[3:0] |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2MULT | 0 | 0 | 0 | 0 | 0 | x1x1 | Xxxx | xxxx |
| 4ADD32 | 0 | 0 | 1 | 0 | 0 | xxxx | x000 | 1100 |
| 4ADD16 | 0 | 0 | 1 | 0 | 1 | xxxx | 1111 | 1000 |
| 4MULT | 0 | 1 | 0 | 0 | 0 | 0000 | Xxxx | xxxx |
| 4MULTSUM | 0 | 1 | 0 | 0 | 1 | 0000 | x000 | x100 |
| 4MULT2SUM | 0 | 1 | 0 | 1 | 0 | 0000 | xx00 | 1111 |
| 4FIR | 0 | 1 | 1 | 0 | 0 | 0000 | 0000 | 0000 |
| CMULT | 0 | 1 | 1 | 1 | 0 | 0000 | 0000 | 1100 |
| CMULT16 | 0 | 1 | 1 | 1 | 1 | 0000 | 0x11 | 0x11 |

Input Muxes (2 new CSMMULT bits)

| Mux / Select | 0 | 1 | 2 | 3 |
| --- | --- | --- | --- | --- |
| mult0-a | i0[15:0] | i0[31:16] | i2[15:0] | 16'h0 |
| mult0-b | i1[15:0] | i1[31:16] | i1[31:8] | 24'h0 |
| mult1-a (mult0-a) | i0[31:16] | i0[15:0] | i2[15:0] | 16'h0 |
| mult1-b (mult0-b) | i1[31:16] | i1[15:0] | i1[31:16] | 16'h0 |
| mult2-a | 16'h0 | i2[31:16] | i0[31:16] | i2[15:0] |
| mult2-b | 16'h0 | i3[31:16] | i3[31:8] | i1[15:0] |
| mult3-a (mult2-a) | 16'h0 | i2[15:0] | i0[15:0] | i2[15:0] |
| mult3-b (mult2-b) | 16'h0 | i3[15:0] | 16'h0 | i1[31:16] |

Note that the mult1 and mult3 a and b operand muxes are derived from the mult0 and mult2 a and b operand muxes respectively. For the 16 bit high low selects, it is useful to note that selecting the high part of the word for mult0 selects the low part of the same word for mult0. This is true for the low bits of all of the select fields, but the high bits are reserved for the more special cases, such as 24*16 multiplies, the FIR opcode, and the CMULT opcode inputs.

Mux input for adders (controls are decoded by opcodes)

| Mux / Select | 0 | 1 | 2 | 3 |
| --- | --- | --- | --- | --- |
| add0-a | i0[31:0] | m0[31:0] | 32'h0 | desp0[31:0] |
| add0-b | i1[31:0] | m1[31:0] | ~m1[31:0]+1 | 32'h0 |
| add1-a | i2[31:0] | m2[31:0] | a2[31:0] | desp1[31:0] |
| add1-b | i3[31:16] | m3[31:0] | 32'h0 | 32'h0 |
| add2-a | a0[31:0] | m0[31:0] | 32'h0 | desp2[31:0] |
| add2-b | a1[31:0] | a0[31:0] | i2[31:0] | 32'h0 |
| add3-a | a2[31:16] | a1[31:0] | 32'h0 | desp3[31:0] |
| add3-b | a2[15:0] | m2[31:0] | i3[31:0] | 32'h0 |

Output Muxes (2 new CSMMULT bits)

| Mux / Select | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| omux0 (mult-h) | m0[31:0] | lsmval | i0[31:0] | i1[31:0] | a2[31:0] | {m0,m1} | a0[31:0] | {a0,a1} |
| omux1 (mult-l) | m2[31:0] | lsmval | i2[31:0] | i3[31:0] | a2[cout] | {m2,m3} | a1[31:0] | a3[31:0] |

### Mult Shift Up (1 new CSMMULT bit)

Selectively causes the output of the multipliers to be shifted up by one bit for normalization. The LSB is then connected to Gnd.

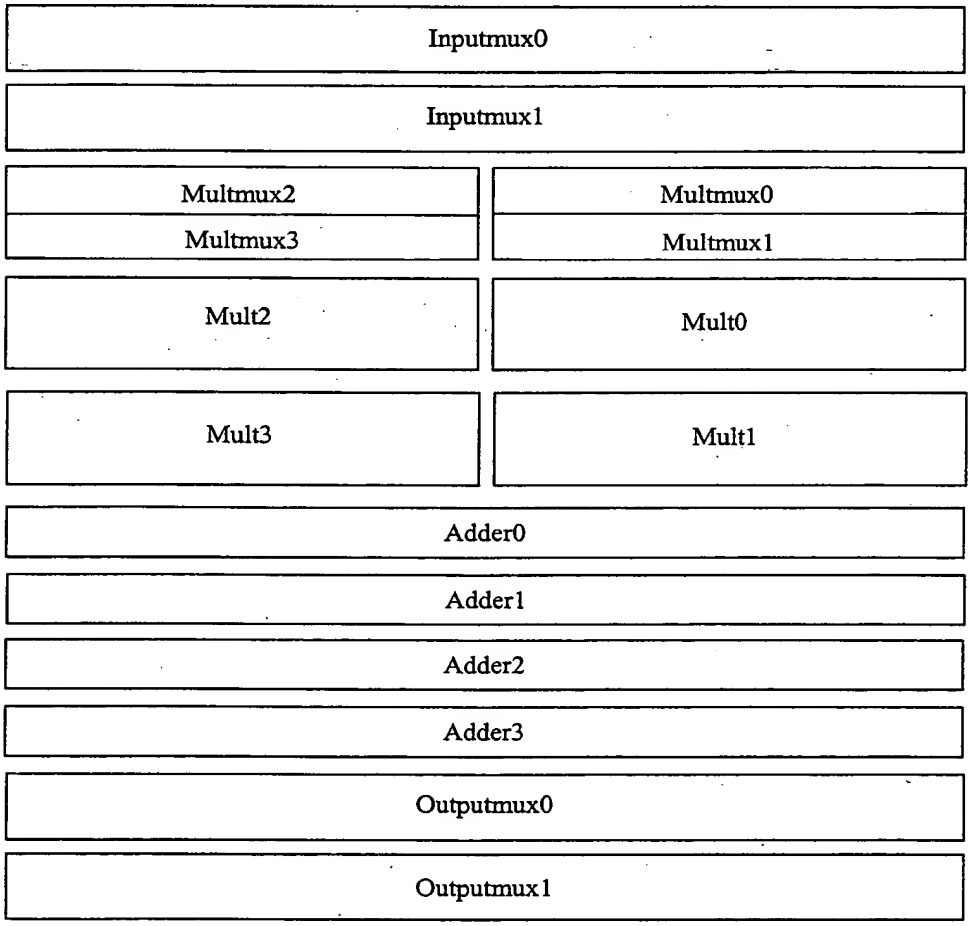
### Adder Shift Down (1 new CSMMULT bit)

Selectively causes the output of the adders to be shifted down by one bit for normalization.

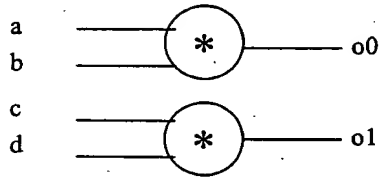
### Input Mux (1 new population on the interconnect input mux)

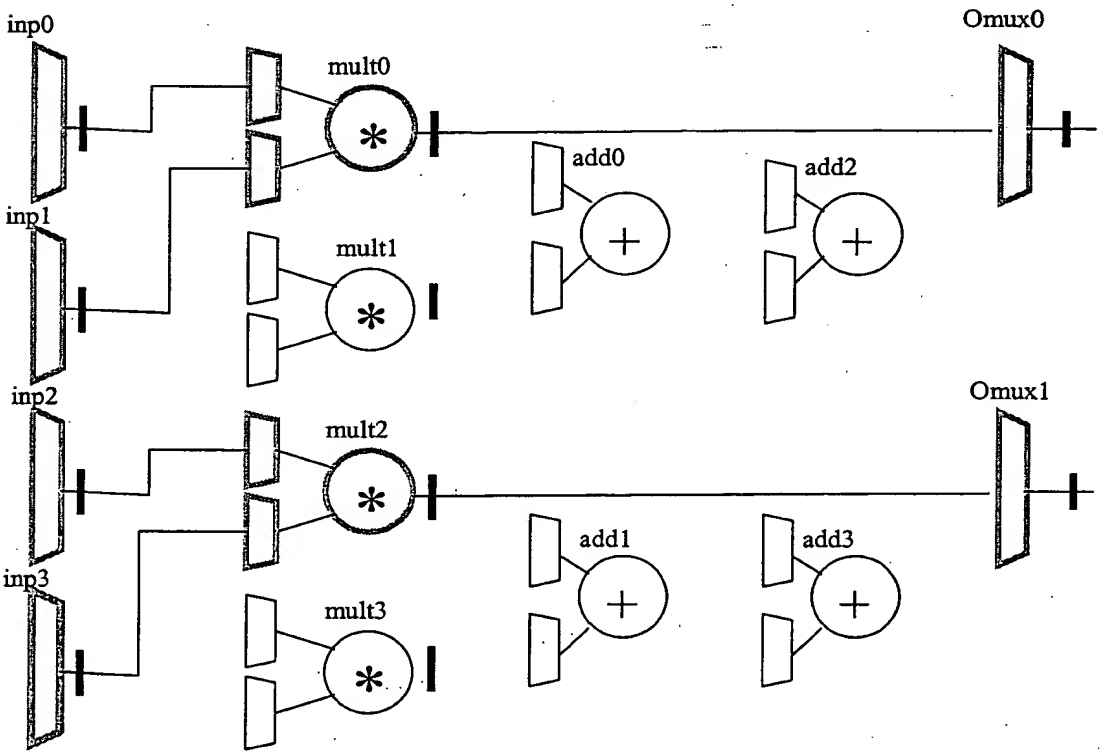
The current input mux supports the constants 0 and –1. It is proposed to add the constant 0x00010001 to allow selective multiplication by 0, 1 and –1.

## 2.2 Layout Floorplan

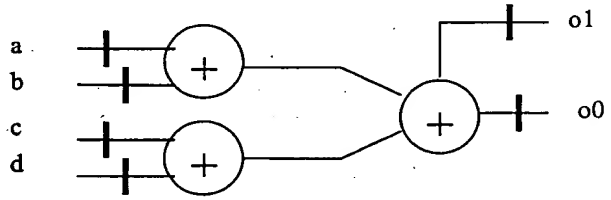


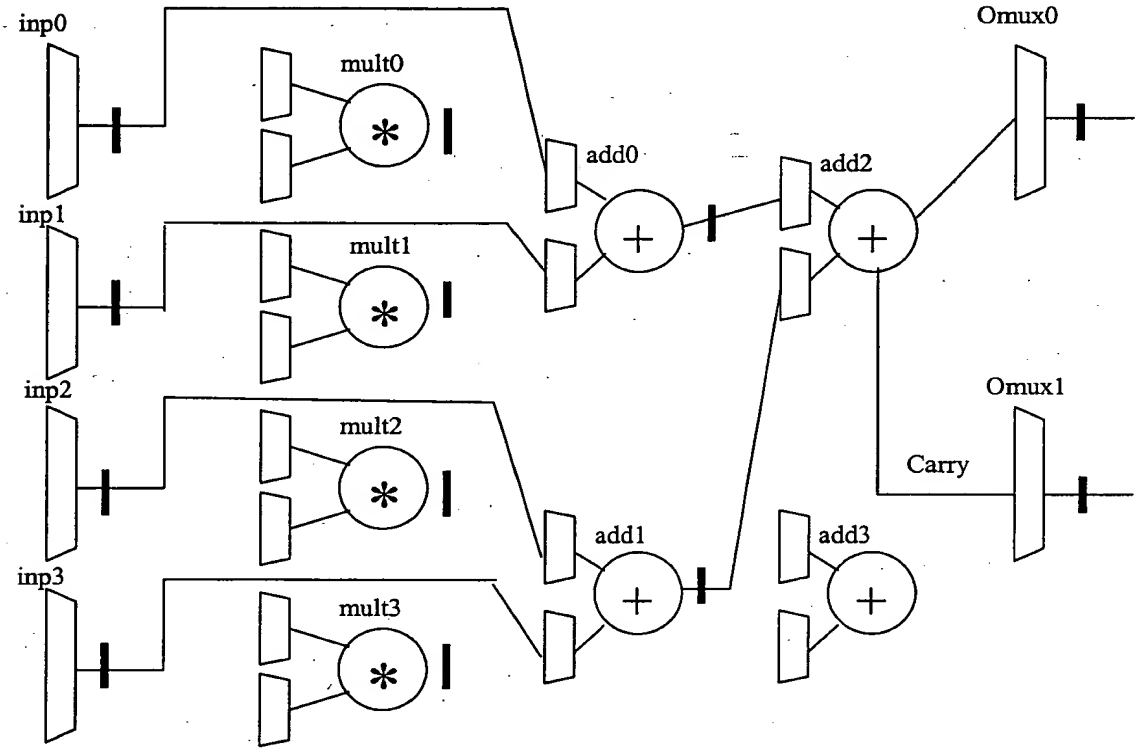
**2MULT – CS2112 Compatible mode 2 independent multipliers**



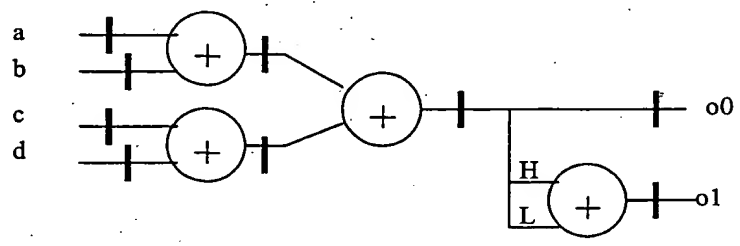



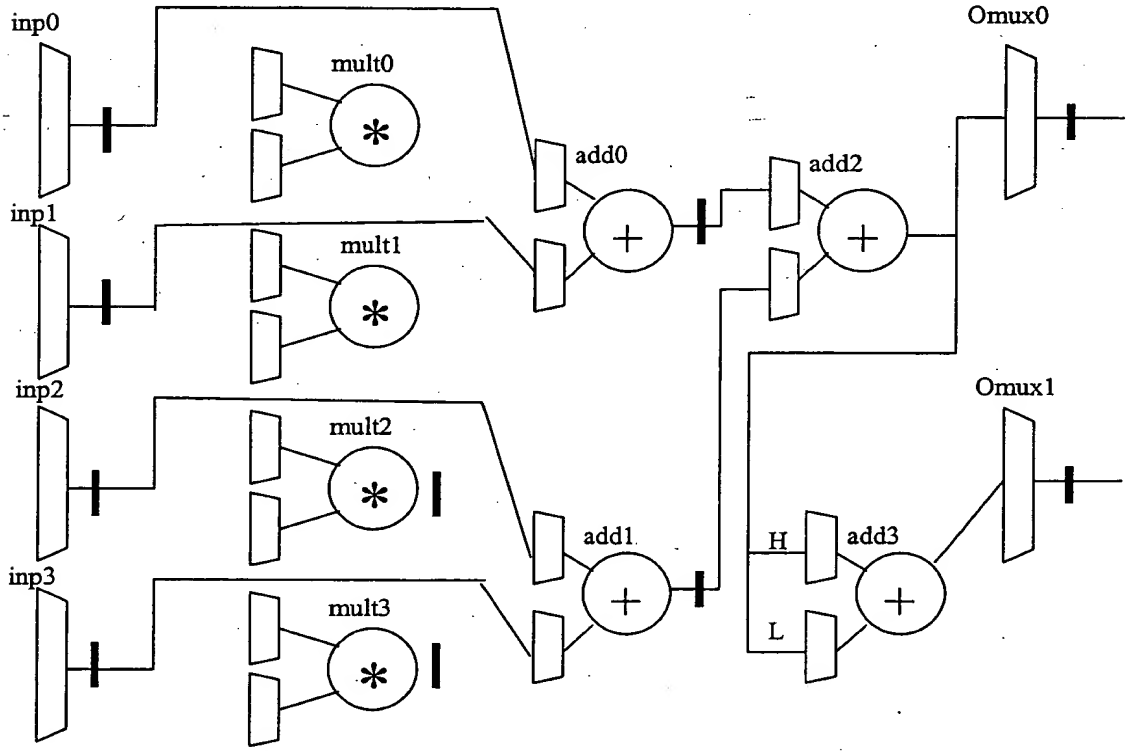
**4ADD32 – Sum of 4 32-bit inputs**



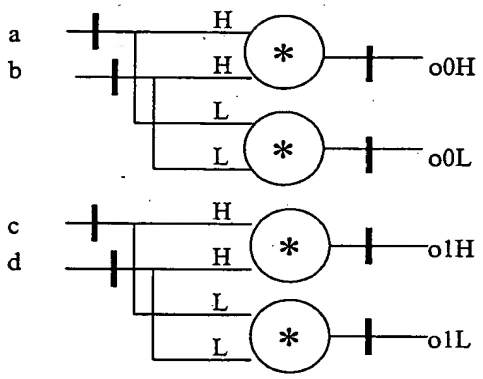



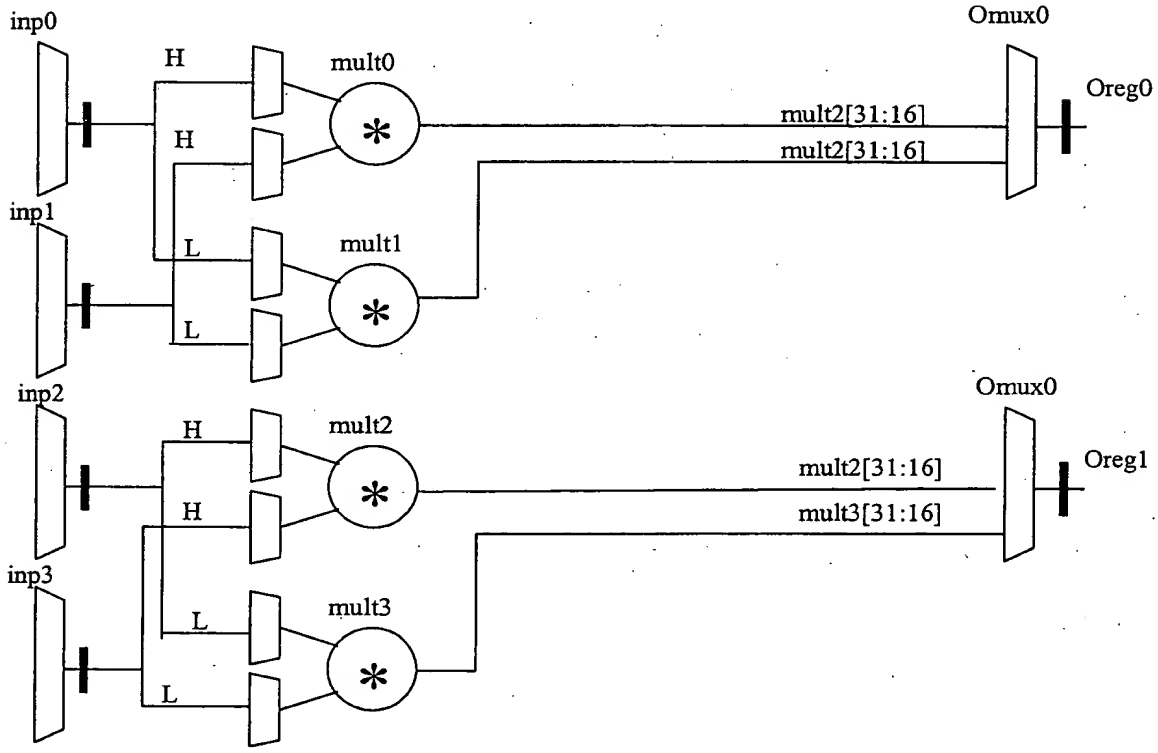
**4ADD16 – Sum of 4 packed 16-bit inputs, sum of upper, lower 16-bits**



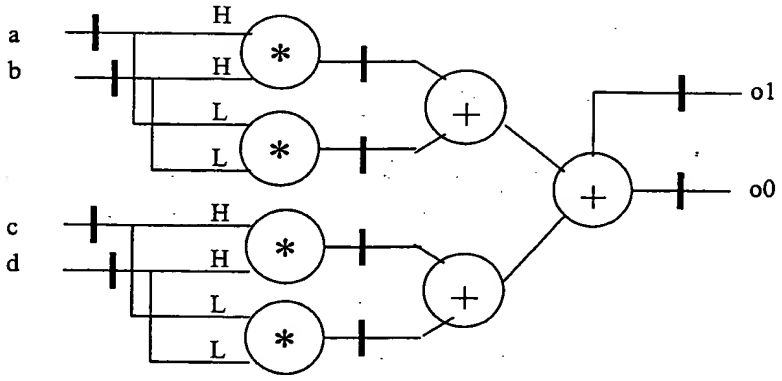



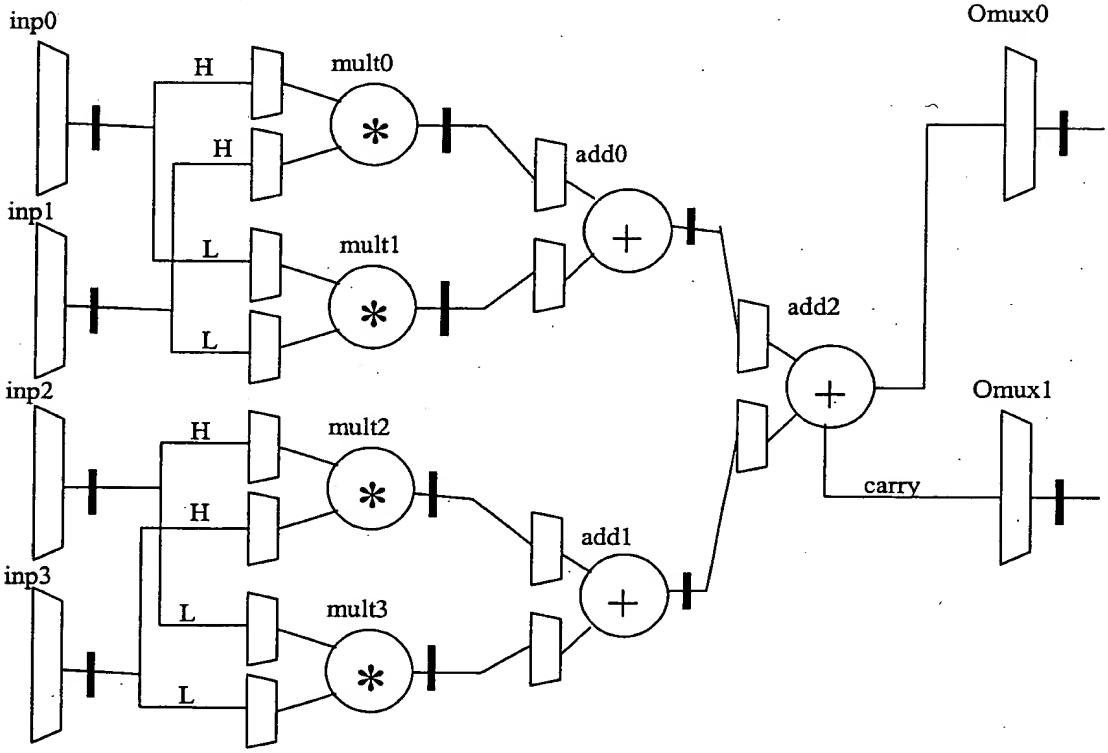
**4MULT – 4 multipliers with pack 16-bit inputs**



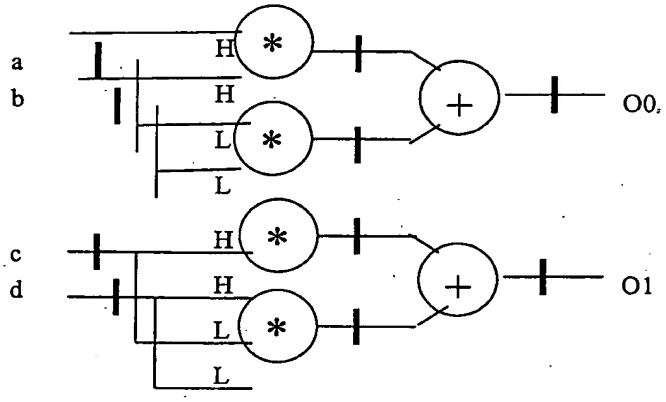



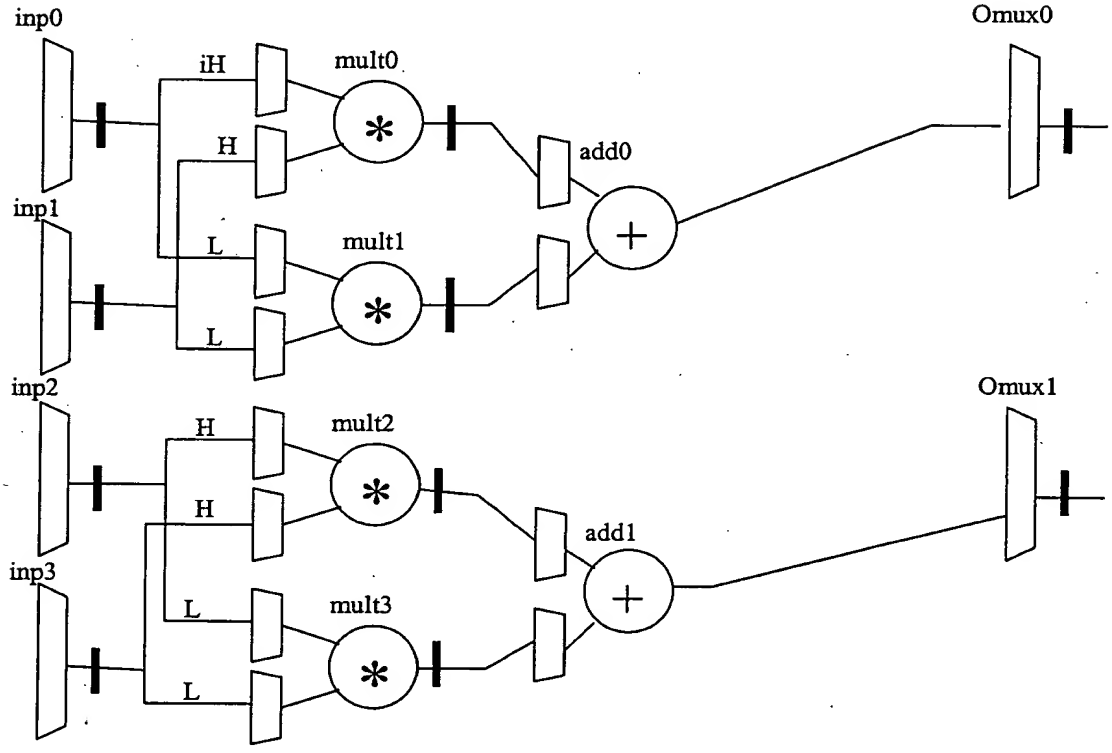
**4MULTSUM – Sum of 4 multipliers**



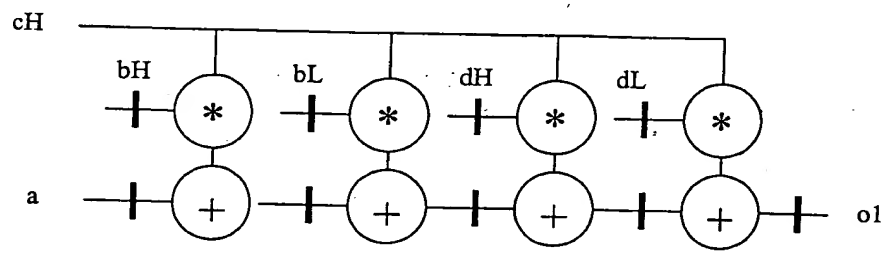



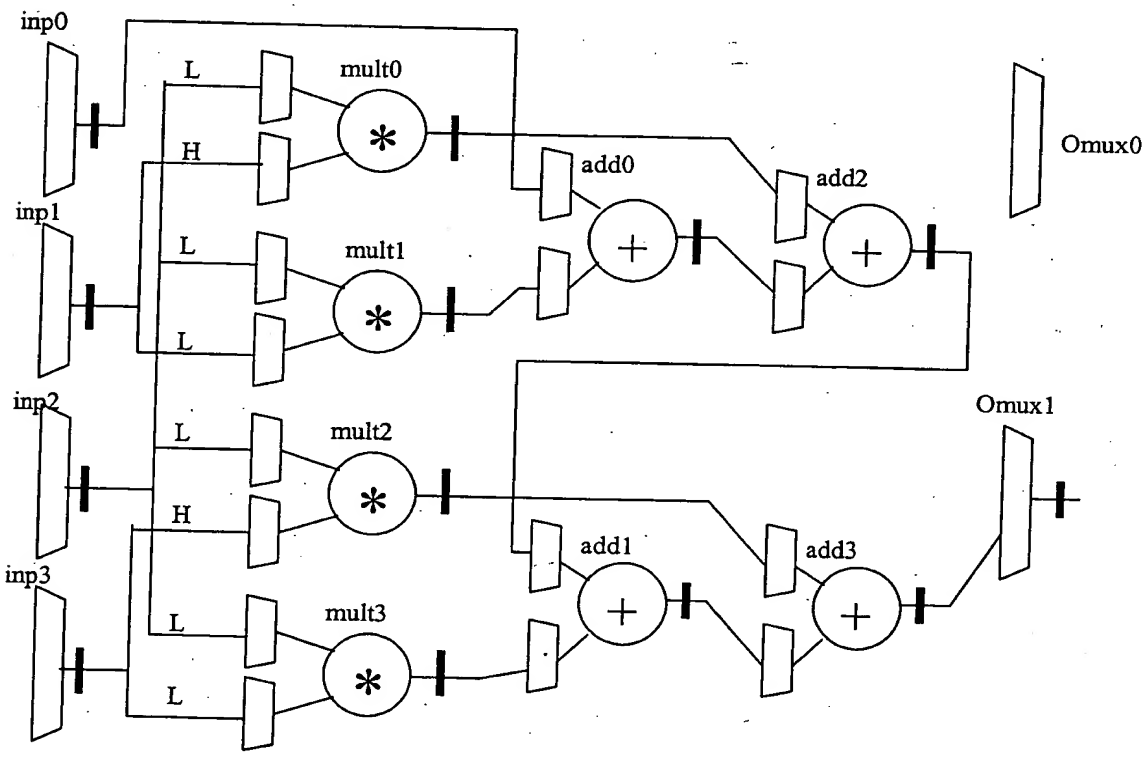
**4MULT2SUM – 2 Sums of 2 multipliers**



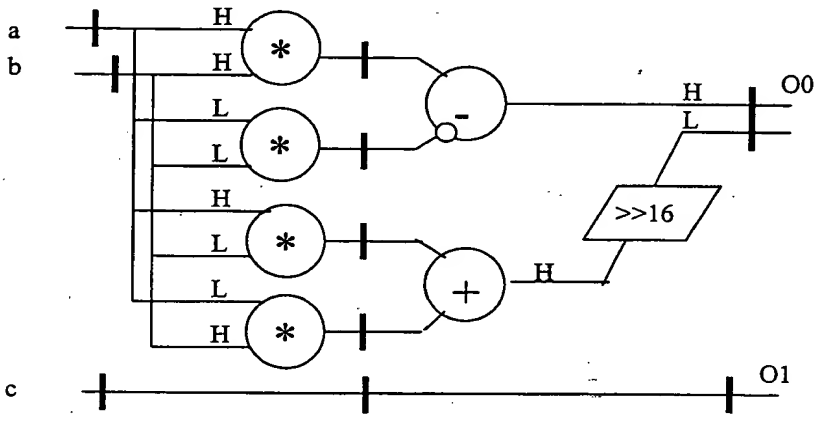



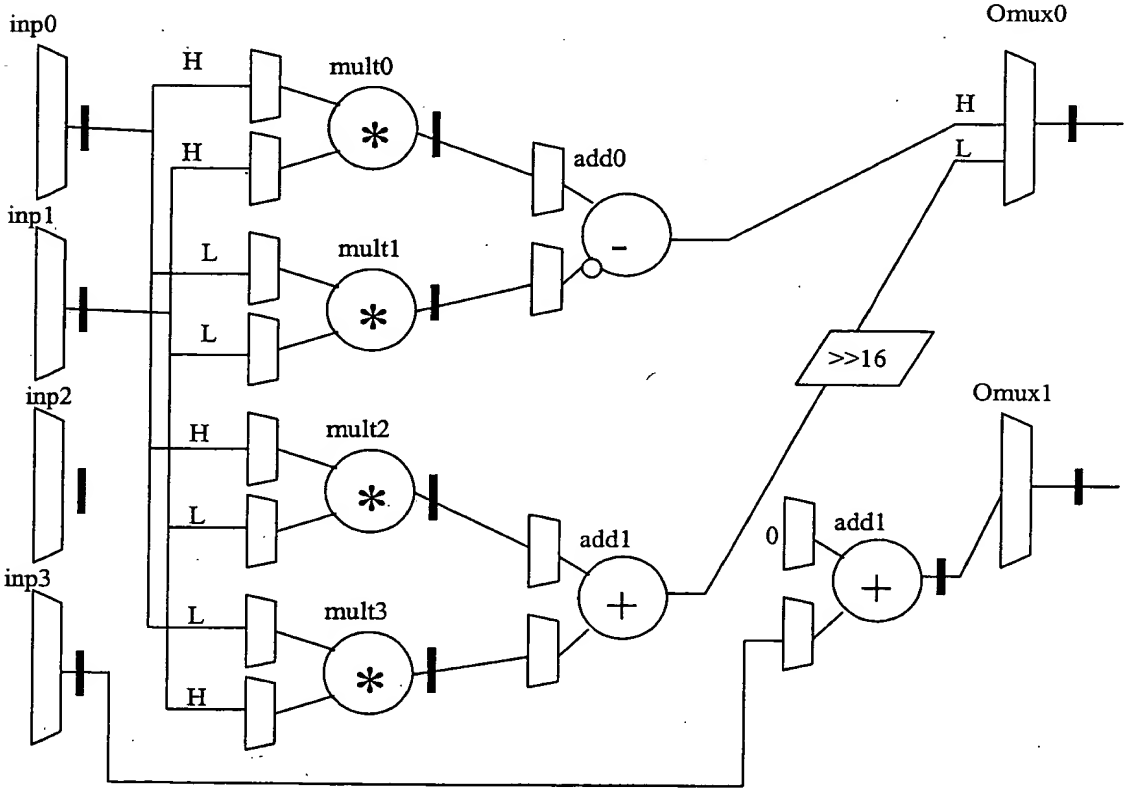
**4FIR - 4 tap FIR filter**



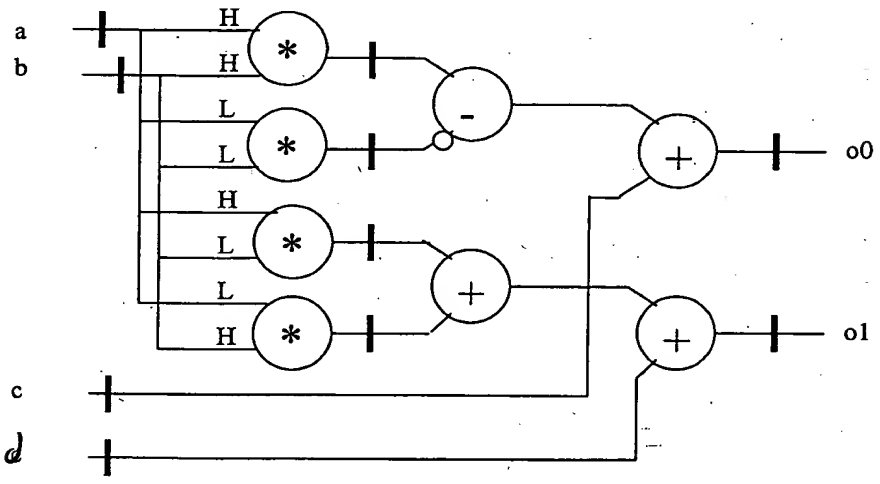



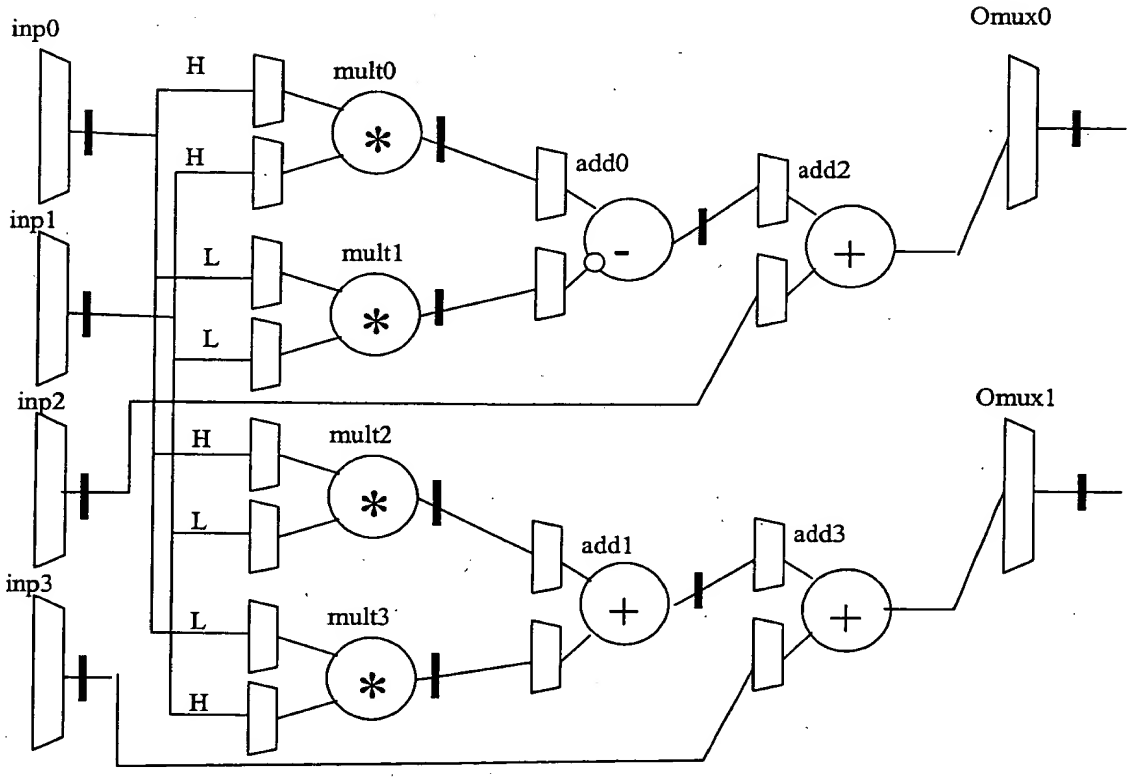
**CMULT16 – Complex Multiplier with 16-Bit Packed data, and indepent delay path. Assumes real part in High 16-bits, imaginary in Low 16-bits**



**CMULT – 32-bit output complex multiply with 32-Bit accumulation input, Assumes real part in High 16-bits, imaginary in Low 16-bits**



## Layout Floorplan for enhanced multiplier in Vermont